面向对象

构造器参数太多怎么办？

用 builder 模式，用在
1、5 个或者 5 个以上的成员变量
2、参数不多，但是在未来，参数会增加

Builder 模式：

属于对象的创建模式，一般有

1. 抽象建造者：一般来说是个接口，包含 1）建造方法，建造部件的方法（不止一个），2）返回产品的方法
2. 具体建造者
3. 导演者，调用具体的建造者，创建产品对象
4. 产品，需要建造的复杂对象

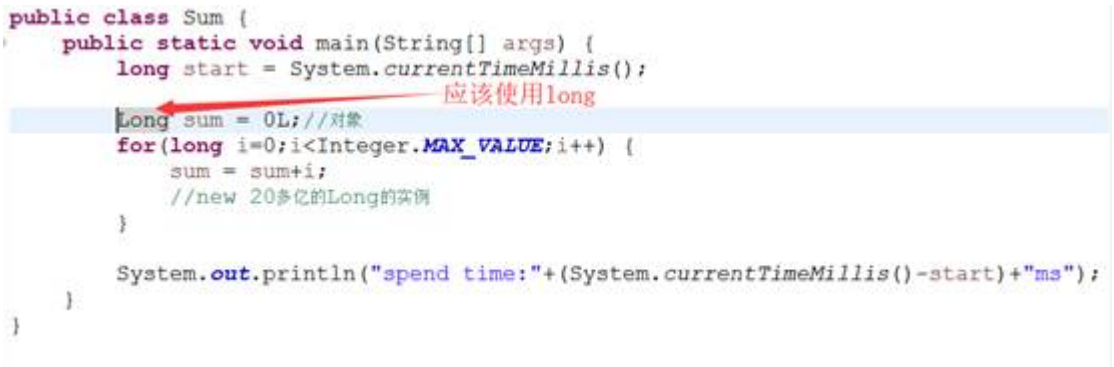
对于客户端，创建导演者和具体建造者，并把具体建造者交给导演者，然后由客户端通知导演者操纵建造者进行产品的创建。
在实际的应用过程中，有时会省略抽象建造者和导演者。

不需要实例化的类应该构造器私有

如，一些工具类提供的都是静态方法，这些类是不应该提供具体的实例的。可以参考 JDK 中的 Arrays。

不要创建不必要的对象

1. 避免无意中创建的对象，如自动装箱




```
public class Sum {
    public static void main(String[] args) {
        long start = System.currentTimeMillis();

        Long sum = 0L;//对象
        for(long i=0;i<Integer.MAX_VALUE;i++) {
            sum = sum+i;
            //new 20多亿的Long的实例
        }

        System.out.println("spend time:"+(System.currentTimeMillis()-start)+"ms");
    }
}
```

2. 可以在类的多个实例之间重用的成员变量，尽量使用 static。

VISIT…

但是，要记住，是不要创建**不必要**的对象，而不是不要创建对象。
对象池要谨慎使用，除非创建的对象是非常昂贵的操作，如数据库的连接，巨型对象等等。

避免使用终结方法

finalizer 方法，jdk 不能保证何时执行，也不能保证一定会执行。如果有确实要释放的资源应该用 try/finally。

使类和成员的可访问性最小化

编写程序和设计架构，最重要的目标之一就是模块之间的解耦。使类和成员的可访问性最小化无疑是有效的途径之一。

使可变性最小化

尽量使类不可变，不可变的类比可变的类更加易于设计、实现和使用，而且更不容易出错，更安全。
常用的手段：
不提供任何可以修改对象状态的方法；
使所有的域都是 final 的。
使所有的域都是私有的。
使用写时复制机制。带来的问题：会导致系统产生大量的对象，而且性能有一定的影响，需要在使用过程中小心权衡。

复合优先于继承

继承容易破坏封装性，而且会使子类的实现依赖于父类。
复合则是在类中增加一个私有域，引用类的一个实例，这样的话就避免了依赖类的具体实现。
只有在子类确实是父类的一个子类型时，才比较适合用继承。

接口优于抽象类

java 是个单继承的，但是类允许实现多个接口。

所以当发生业务变化时，新增接口，并且需要进行业务变化的类现新接口即可。但是抽象类有可能导致不需要变化的类也不得不实现新增的业务方法。
在 JDK 里常用的一种设计方法是：定义一个接口，声明一个抽象的骨架类实现接口，骨架类类实现通用的方法，而实际的业务类可以同时实现接口又继承骨架类，也可以只实现接口。
如 HashSet 实现了 **implements** Set 接口 但是又 **extends** 类 AbstractSet，而 AbstractSet 本身也实现了 Set 接口。其他如 Map，List 都是这样的设计的。

方法

可变参数要谨慎使用

可变参数是允许传 0 个参数的
如果是参数个数在 1~多个之间的时候，要做单独的业务控制。
看代码

返回零长度的数组或集合，不要返回 null

方法的结果返回 null，会导致调用方的要单独处理为 null 的情况。返回零长度，调用方可以统一处理，如使用 foreach 即可。
JDK 中也为我们提供了 Collections.***EMPTY_LIST*** 这样的零长度集合

优先使用标准的异常

要尽量追求代码的重用，同时减少类加载的数目，提高类装载的性能。
常用的异常：
IlegalAraumentException -- 调用者传递的参数不合适
IllegalStateException – 接收的对象状态不对，
NullPoint
UnsupportedOperationException –不支持的操作

通用程序设计

用枚举代替 int 常量

声明的一个枚举本质就是一个类，每个具体的枚举值就是这个枚举类的实例。
枚举更多作用，看代码。

将局部变量的作用域最小化

1. 在第一次使用的地方进行声明
2. 局部变量都是要自行初始化，初始化条件不满足，就不要声明

最小化的好处，减小局部变量表的大小，提示性能；同时避免局部变量过早声明导致不正确的使用。

精确计算，避免使用 float 和 double

可以使用 int 或者 long 以及 BigDecimal

当心字符串连接的性能

参考代码 com.xiangxue.ch04.StringUnion15.Test。
在存在大量字符串拼接或者大型字符串拼接的时候，尽量使用 StringBuilder 和 StringBuffer

控制方法的大小